Convolutional Neural Networks Explained from Scratch

Venchislaw

Unemployed bruh

Author Note

All sources used are linked at the end of the paper.

venchislavcodes@gmail.com

Abstract

Convolutional Neural Networks are widely used in practical projects and are found efficient and simple. However most of the people using this architecture do not know how it works under the hood, or have a shallow understanding. This paper is not a research or some new method introduction. It is a simple, beginner-friendly guide to convolutional neural networks.

This paper is written to clarify confusions about Convolutional Neural Networks!

Hope you enjoy it! GitHub - https://github.com/Venchislaw

Why CNN?

UseCase

These networks are commonly used for computer vision problems.

These problems are very specific and differ from structured tabular data ones.

Our images usually vary, so we need to keep our eye on the geometric structure of the data passed, and do it in an efficient way.

Why not MLP?

Ordinary Fully Connected MLP fails both expectations, as data is “flattened” (loss of geometric structure) and the network uses dozens of parameters.

Solution.

To understand CNNs we need to understand how we, human beings see.

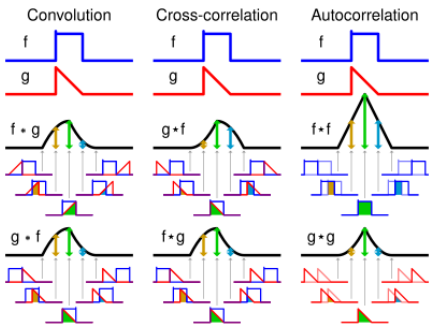
One guess says that our brain detects some “significant” features of the object to detect it. Our brain detects edges, corners, line thickness, colors and many other properties to make the final decision. Our brain learned it somehow, learned how to detect these features.

Convolutional Neural Network structure basically aims to learn the model to detect all these properties from the image. It assumes that we apply some “filters” to the image, so We detect edges and other useful information. This Neural Network learns the filters to perform the most efficient feature extraction.

What is CNN?

Convolution

Operation of convolution plays an important role in our network, as the name suggests. There is a math background of this operation to cover, so let’s start with simple things. In math it is an operation on two functions that yields a third function,
such that the output function represents how much the “shape” of the function is changed by the other.



Pic.1 Convolution. Source: WKP[1]

Good intuition on what is a convolution without boring math is a statistics representation.

In Stats it is represented as a **weighted moving average.** But what about images?

We can apply operation of convolution to the images as long as we assume that our

Images are represented with Matrices.

Convolution on the images with various “filters” can give us many interesting outputs.

Aha! We’ve already used the word “filter” in this paper. Furthermore, these filters work

As a PhotoShop. With the right filter we can blur the image or detect edges or sharpen it

Etc. The best thing about it all is that this filter is relatively small and glides across our

Image. So, It takes less parameters, keeps our image’s geometric structure and neglects

Feature position (yeah, in CNNs position of the object on the image does not matter), so

What is the catch? The catch is… It is not called Convolution.

Is it confusing? Yes!

Is it for real? Absolutely!

The trick is - our convolution operation **flips** the second array (filter in this case), but

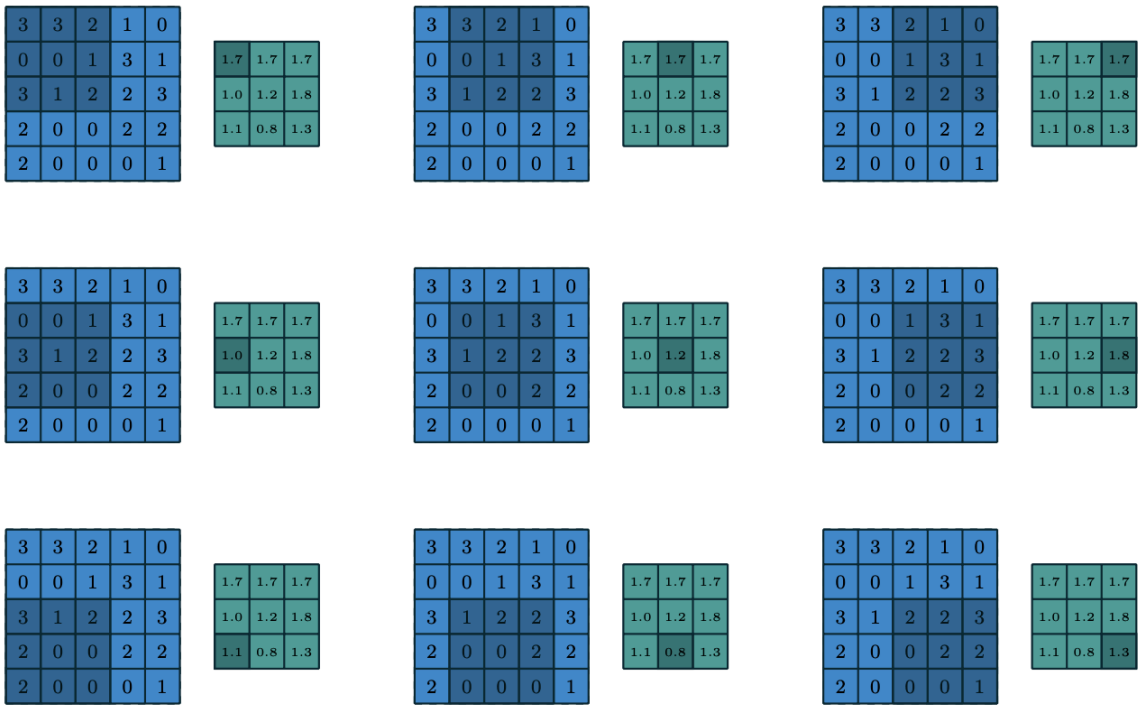
We didn’t do it. It may sound funny and pointless, but in mathematics it plays an

Important part, so we can’t neglect it.

But what is the name of this operator if it’s not a convolution?

Cross-Correlation! Does absolutely the same thing for images, but without flip.

It is depicted in Pic.1.



Pic.2 Convolution on the Image. Source: PWC[2]

Nice intuition is a dot-product between kernel values and input image part.

Convolution Layer

Let's try to implement a convolutional layer. Let's start with a **grayscale** image.

The most famous example of such grayscale images is the MNIST dataset.

Pic.3 MNIST. Source: WKP[1]

It's just a set of black and white images, each of size 28x28 pixels with pixel values Encoded in range from 0(black) to 255(white). Convolution operation on the layer Is identical to the one on the Pic.2, but we also add a **bias** value.

Convolution operation on the image yields the other image of size:

$$W_o = \frac{W_i + 2p - f}{s} + 1 \quad (1)$$

$$H_o = \frac{H_i + 2p - f}{s} + 1 \quad (2)$$

Let's clarify what these fancy letters are.

W_i - width of the input

W_o - width of the input

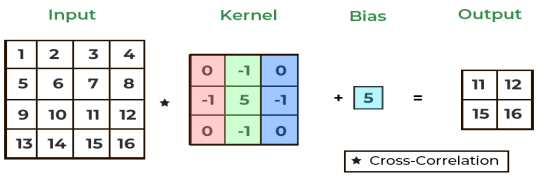
H_i - height of the input

H_o - height of the input

p - padding (we'll clarify it below)

f - filter size (size of our filter)

s - strides (we'll also clarify it)



Pic.3 CNN. Source: GFG[3]

The output in this case is 4x4x1 image. However in Convolutional Neural Nets we can Apply many filters in one layer. In this way we'll produce 4x4xf_n image.

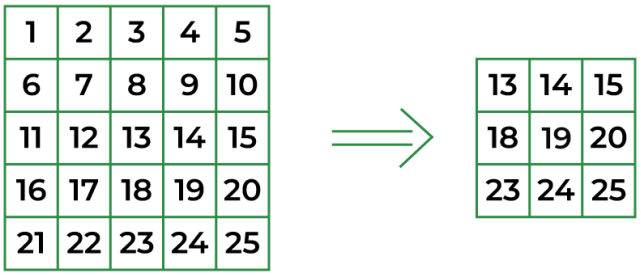
Previously in Notations I mentioned **Padding** and **Strides.** Let's find out what these are!

Padding

If you think deeper about what is happening in convolution operation, you'll understand, that we pay less attention to the pixels on the edges, because we glide through them once. That's why we add padding - a "frame" of 0s to process edges more. Also this method Can keep our image size bigger or at least the same. Yep, formulas (1) and (2) suggest, That our image size shrinks, but with the right **padding** value we can keep the image Size the same. We can distinguish three types of padding:

Valid

Means no padding(p=0). Pretty straight forward.



Pic.4 Valid Padding. Source: GFG[3]

Same

Preserves image size from shrinkage. Padding value is calculated as:

$$p = \frac{f-1}{2} \quad (3)$$

If you are curious about how we came up with this equation:

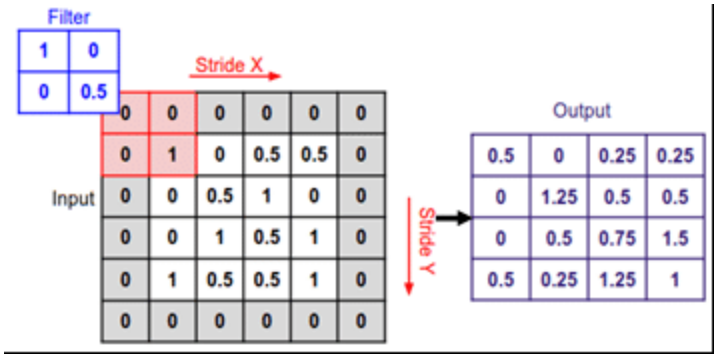
Let's ignore strides (s=1)

$$n + 2p - f + 1 = n$$

$$2p - f + 1 = 0$$

$$2p = f - 1$$

$$p = \frac{f-1}{2}$$



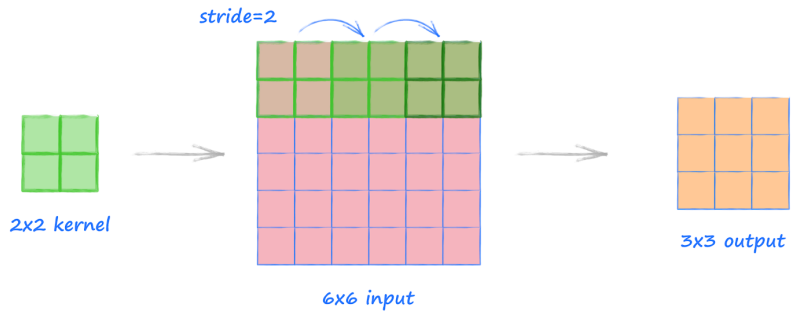
Pic.4 Same Padding. Source: RSG[4]

Custom

Set padding as a hyperparameter.

Strides

Strides are used to control the output image size. Roughly stride is a step size for our sliding window. If we set s=2, our output image size will be twice smaller (with p=0)



Pic.5 Strides. Source: MYN[5]

Convolution On the colorful Image

Thankfully we see the world in colors. Representation of such images is a 3D Matrix

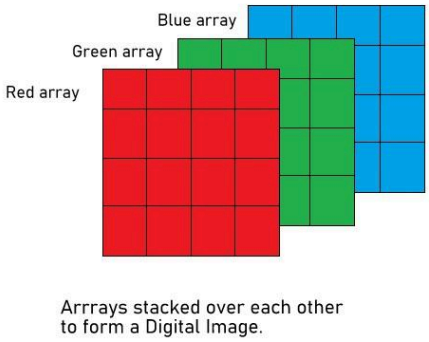
$$W \times H \times C$$

W and H are width and height, respectively, while C is a number of channels.

For colorful images we usually have 3 color Channels: RGB (Red Green Blue) see Pic.6

The solution is really simple. For 3D images we use 3D filters. Each filter chanel slides Across the image channel so we produce 2D Matrix.

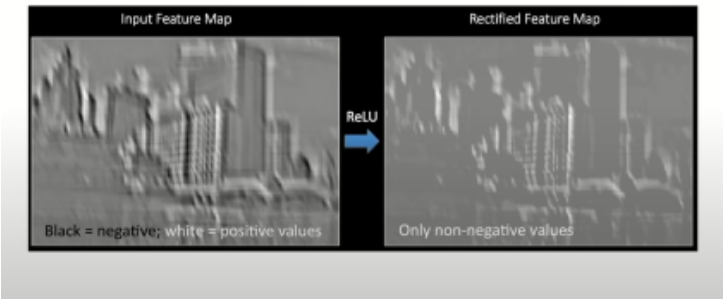
It may sound like a problem, but it is not. We stack our filters, so it's good for us.



Pic.6 Color Channels. Source: GFG[3]

Finally, as we're working with Neural Networks we apply **activation function** to our Matrices (elementwise). Usually ReLU is used as an activation function.

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



Pic.7 ReLU Non-Linearity. Source: MIT[6]

Layers In Convolutional Neural Network

Convolutional Neural Networks do not only use Conv Layers. We have few main Types of Layers:

- Convolutional Layer
- Pooling (explained below)
- Dense (fully connected layer from MLP)
- Flatten (explained below)

Okay, what is Pooling?

Pooling

Pooling Layer reduces the size of our image, preserving useful features.

It is used to speed up training and save only important features.

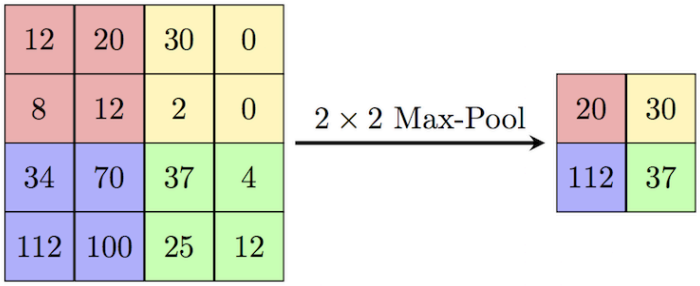
It's 2 main Pooling Types out there:

MaxPooling

AveragePooling

Max Pooling

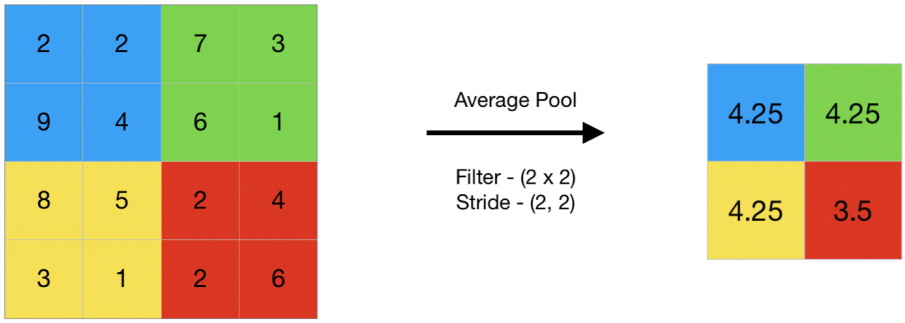
Max Pooling slides the window (pool) across the image taking the maximum Color value from this pool.



Pic.8 Max Pooling. Source: PWC[2]

Average Pooling

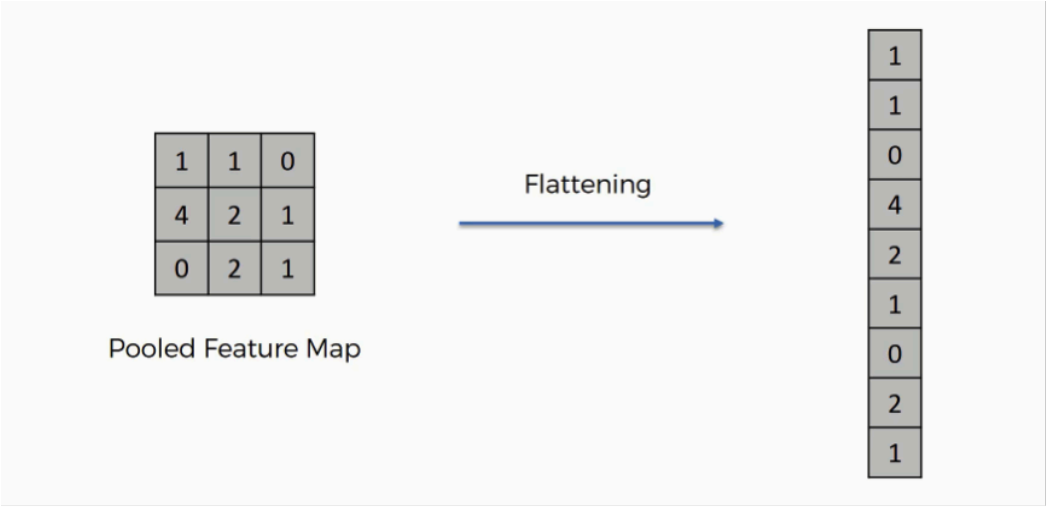
Average Pooling slides the window (pool) across the image taking the average Color value from this pool.



Pic.9 Average Pooling. Source: GFG[3]

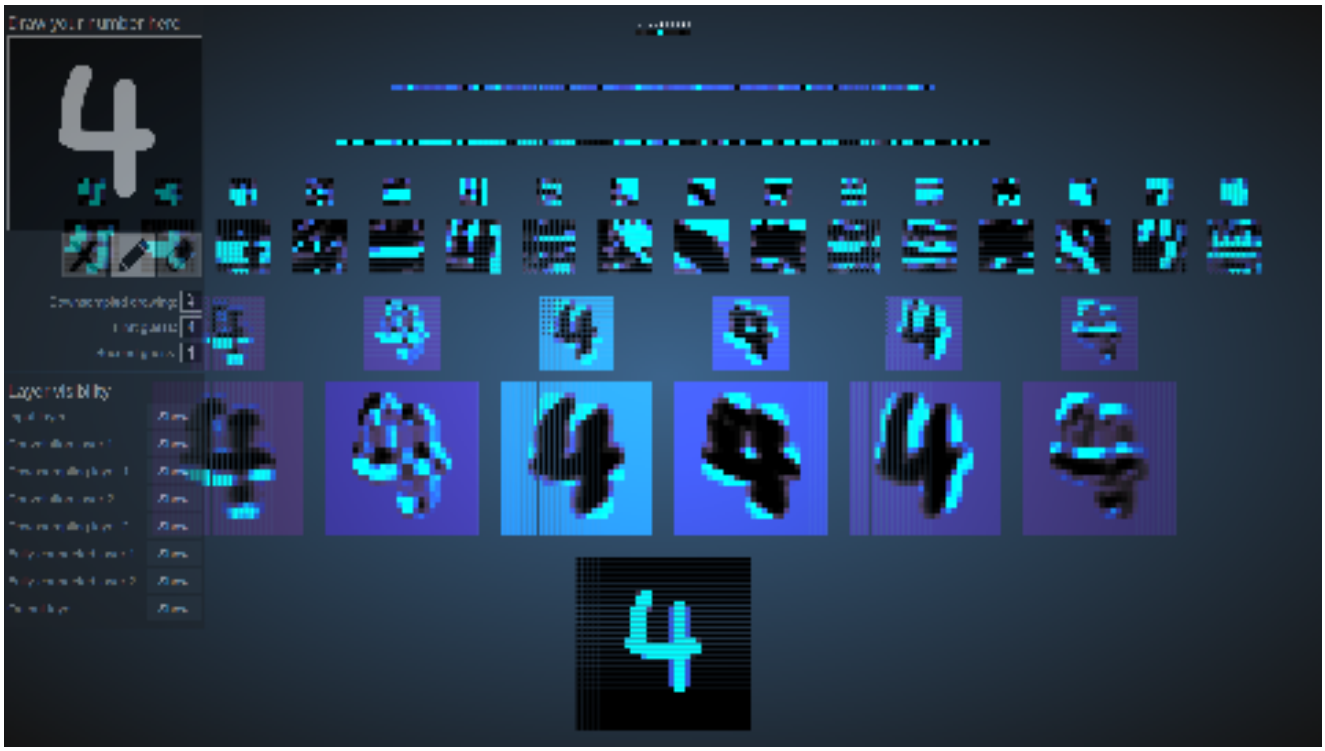
Flatten Layer

Flattens matrix, so we can pass it to the Dense layer.



Pic.9 Flatten. Source: SDS[7]

Final Network Structure:



Pic.10 Final Network Structure. Source: AWH[8]

PART II

MATH

Math Can get confusing, so it's important to clarify everything.

Syllabus of the Chapter:

- Forward Propagation of Convolutional Neural Net
- Backward Propagation of Convolutional Neural Net

Moving Forward

Forward Propagation of Convolutional Neural Network is fairly simple, so without further ado, let's start! The most interesting layer out of all described above is a convolutional layer.
By the way, we've already noticed that the operation of Cross-correlation is performed instead of the Convolution operation.

Convolutional Layer under the hood

Each i_{th} filter of the layer performs the following calculation:

$$Y_i = B_i + X_1 \star K_{11} + X_2 \star K_{12} + \ + X_n \star K_{1n}$$

In this and later equations we're gonna use the following notation:

n - number of input channels

d - number of filters (number of output channels)

As you can see, we add up 2D matrices (as each channel convolution with 1 filter outputs 2D matrix).

We perform this calculation on all the filters:

$$Y_1 = B_1 + X_1 \star K_{11} + X_2 \star K_{12} + \ + X_n \star K_{1n}$$

$$Y_2 = B_2 + X_1 \star K_{21} + X_2 \star K_{22} + \ + X_n \star K_{2n}$$

$$\vdots$$

$$Y_d = B_d + X_1 \star K_{d1} + X_2 \star K_{d2} + \ + X_n \star K_{dn}$$

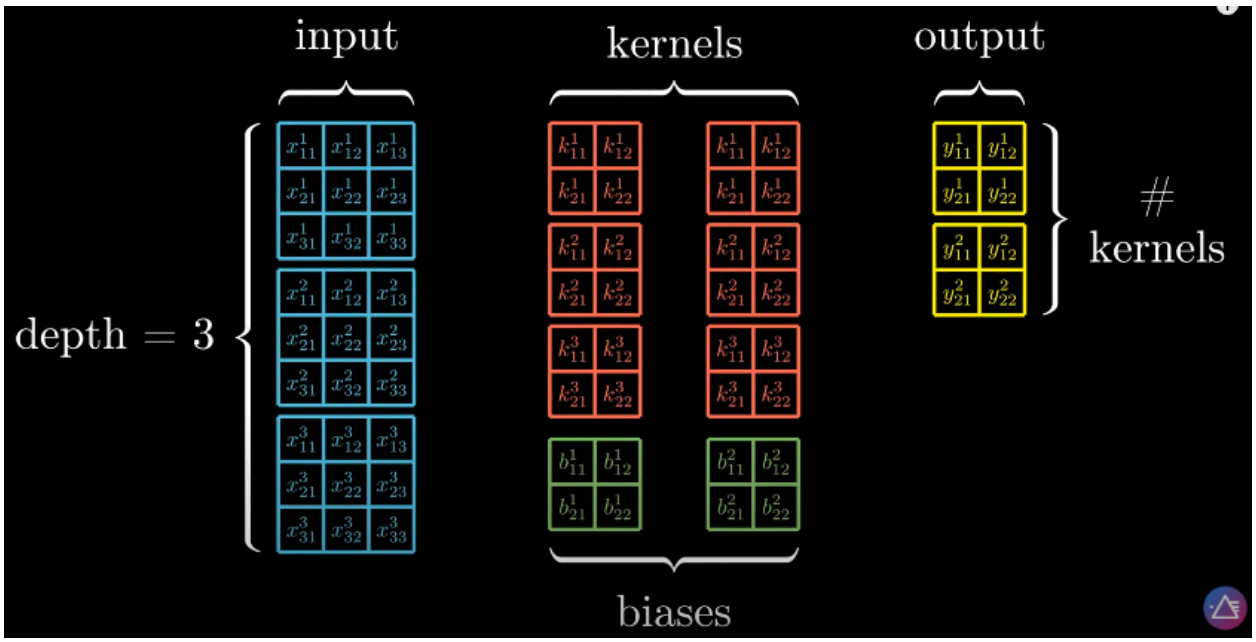
Note: X_n denotes n_{th} channel of the input image

More compact form to write it is the following notation with a sum:

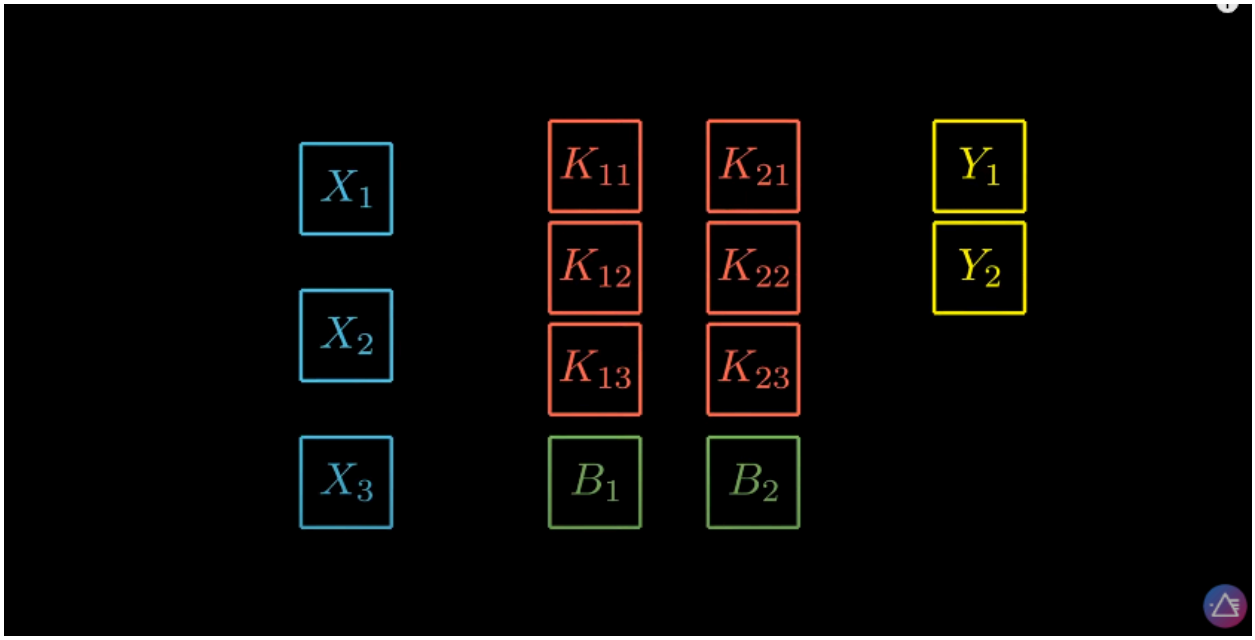
$$Y_i = B_i + \sum_{j=1}^{n} X_j \star K_{ij}$$

Where: $i = 1 \ldots d$

Nice! We have a general form for channel cross-correlation.



Pic.11 Notations clarified. Source: TIC[9]



Pic.11 The same notation, less details. Source: TIC[9]

Moonwalking

I decided to give this part a funny name because this part is traumatizing and I decided to cheer you up before this nightmare. Okay, I'm just kidding, but stay tuned, because it's easy to lose something.

I'm not going to cover backprop of ordinary Dense layers, because It partially relates to the topic.

In convolutional Layer (which is the most confusing among them all) we need to find the following partial derivatives:

$$\frac{\partial L}{\partial k_{ij}} \frac{\partial L}{\partial B_i} \frac{\partial L}{\partial X j}$$

I hope you know that we use the **chain rule** here, and that $\frac{\partial L}{\partial k_{ij}} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial k_{ij}}$

Let's start breaking it down!

$\frac{\partial L}{\partial k_{ij}}$

Alright, we want to find out how much the j_{th} channel of the i_{th} kernel affects our loss, so we can backpropagate our loss. From chain rule we know it should be:

$$\frac{\partial L}{\partial k_{ij}} = \frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial k_{ij}}$$

$\frac{\partial L}{\partial Y_i}$ is pretty straight forward, it's just a partial derivative of loss (it's not related to the topic of convolution). The most interesting part is how much change in k_{ij} affects Y_i

To find it out we need to write down equation of Y_i.

$$Y_i = B_i + \sum_{j=1}^{n} X_j \star K_{ij}$$

Let's reveal the whole equation:

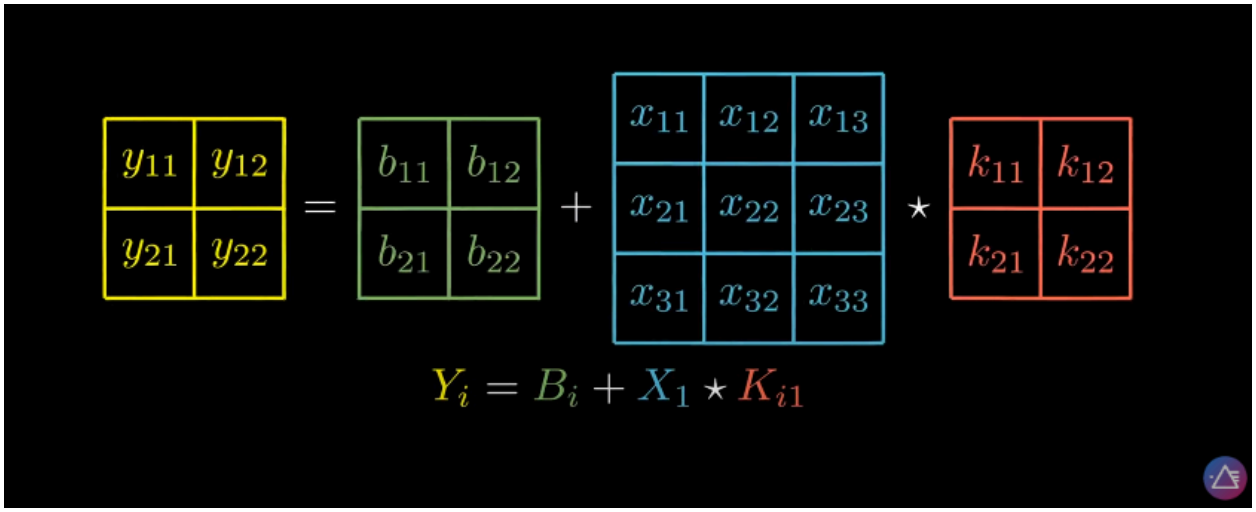
$$Y_i = B_i + X_1 \star K_{i1} + X_2 \star K_{i2} + \ldots + X_n \star K_{in}$$

This is the output of 1 i_{th} kernel for n channels.

Let's work with simplified version for now, where we have only 1 channel ($n = 1$)

Such images are called "grayscale images" (see Pic.3).

$$Y_i = B_i + X_1 \star K_{i1}$$



Pic.12 Calculation for the kernel i with 1 channel. Source: TIC[9]

Let's show Y_i elements calculation:

$$y_{11} = b_{11} + x_{11}k_{11} + x_{12}k_{12} + x_{21}k_{21} + x_{22}k_{22}$$

$$y_{12} = b_{12} + x_{12}k_{11} + x_{13}k_{12} + x_{22}k_{21} + x_{23}k_{22}$$

$$y_{21} = b_{21} + x_{21}k_{11} + x_{22}k_{12} + x_{31}k_{21} + x_{32}k_{22}$$

$$y_{22} = b_{22} + x_{22}k_{11} + x_{23}k_{12} + x_{32}k_{21} + x_{33}k_{22}$$

(4)

From these equations we can derive any Y_i w.r.t any k_{ij}

$$\frac{\partial Y_i}{\partial k_{11}} = x_{11} + x_{12} + x_{21} + x_{22}$$

As we need to find $\frac{\partial L}{\partial k_{ij}}$ and we know it's $\frac{\partial L}{\partial Y_i}\frac{\partial Y_i}{\partial k_{ij}}$, we come up with the following calculation:

$$\frac{\partial L}{\partial k_{11}} = (x_{11} + x_{12} + x_{21} + x_{22})\frac{\partial L}{\partial Y_i}$$

Oh… Say the line…

This operation is a cross-correlation between our input channel and $\frac{\partial L}{\partial Y_i}$

Now we have a formula for one channel of the image:

$$\boxed{\frac{\partial L}{\partial k_{ij}} = X_j \star \frac{\partial L}{\partial Y_i}}$$

$$\frac{\partial L}{\partial B_i}$$

Okay, As we've already dived into the details of what's going on, let's work on biases.

Again, let's work with one channel for now. Check Equation block (4).

From there we can see that $\frac{\partial Y_i}{\partial b_{ij}} = 1$.

That is to say:

$$\boxed{\frac{\partial L}{\partial B_i} = \frac{\partial L}{\partial Y_i}}$$

Tbh, this explanation is short (maybe even too short), but it's because I've already shown everything in loss w.r.t kernel part (in equation block(4)).

$$\frac{\partial L}{\partial X_j}$$

We need this derivative for previous layers derivation.

Okay, thankfully it's our last derivation. Once again, check out equation block (4).

I highly recommend you to derive $\frac{\partial Y_i}{\partial x_{ij}}$

It's time consuming, but I highly recommend you to do it by hand.

Once you've done this, you will see the following picture:

$$\frac{\partial Y_i}{\partial x_{11}} = k_{11}$$

$$\frac{\partial Y_i}{\partial x_{12}} = k_{11} + k_{12}$$

$$\frac{\partial Y_i}{\partial x_{13}} = k_{12}$$

$$\frac{\partial Y_i}{\partial x_{21}} = k_{11} + k_{21}$$

$$\frac{\partial Y_i}{\partial x_{22}} = k_{11} + k_{12} + k_{21} + k_{22}$$

$$\frac{\partial Y_i}{\partial x_{23}} = k_{12} + k_{22}$$

$$\frac{\partial Y_i}{\partial x_{31}} = k_{21}$$

$$\frac{\partial Y_i}{\partial x_{32}} = k_{22} + k_{21}$$

$$\frac{\partial Y_i}{\partial x_{33}} = k_{22}$$

Oh… Equations have various numbers of terms. Hmm… Is there any pattern?

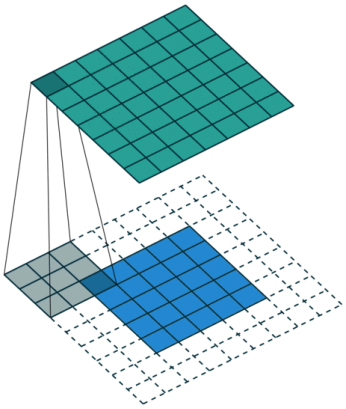
Hell yeah there is! Let's combine it with our $\frac{\partial L}{\partial Y_i}$:

$$\frac{\partial L}{\partial x_{22}} = (k_{11} + k_{12} + k_{21} + k_{22})\frac{\partial L}{\partial Y_i}$$

This is a **Full Convolution** between kernel and loss derived w.r.t output channel.

Okay, In part I we've discussed **Padding.**

Full Convolution (not cross-correlation) is a convolution with such padding that We start with the top left edge of our matrix in the bottom right position of the filter:



Pic.13 First step of Full convolution. Source: Spatial[10]

That said:

$$\frac{\partial L}{\partial X_j} = \sum_{i=1}^{d} \frac{\partial L}{\partial Y_i} *_{full} k_{ij}$$

Where $j = 1 \dots n$

Notice: We have a summation here, because the same X_j is used in d kernels.

Oh My God! We did it! We derived all we needed! However, that's not it.

In this paper I mentioned Pooling layers. These layers also have an effect on our backpropagation.

And what about Padding? Strides? Let's clarify it all!

Padding

Exhale, padding has no influence on our loss function. It is designed only to tune output size.

Strides

They have influence, but I'm too tired to explain it here.

Check out MMK[11] in resources.

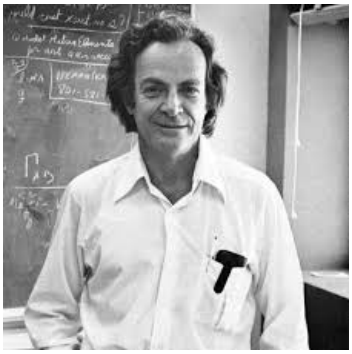
Pooling

In case of max pooling we map calculated gradient to the max item in the pool over the original image. Average pooling fills all items with gradient given (as all pool items had the same influence on the Loss).

Hoooh… It took me 10 Hours of writing, researching, and calculating…

Damn, I'm super duper tired! See additional sources in References.

Thanks for reading this paper!



"What I can not create, I do not understand" - R.Feynman (1988)

References

[1] WikiPedia:

- Convolution [https://en.wikipedia.org/wiki/Convolution]
- MNIST database [https://en.wikipedia.org/wiki/MNIST_database]

[2] Papers With Code:

- Convolution [https://paperswithcode.com/method/convolution]
- MaxPooling [https://paperswithcode.com/method/max-pooling]

[3] Geeks For Geeks:

- CNN [https://www.geeksforgeeks.org/apply-a-2d-convolution-operation-in-pytorch/]
- Valid Padding [https://shorturl.at/CQH9b]
- RGB [https://www.geeksforgeeks.org/matlab-rgb-image-representation/]
- AVG Pooling [https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/]

[4] Researcher Gate:

- Same Padding [https://shorturl.at/28VxO]

[5] Make Your Neural Network:

- Strides [https://shorturl.at/iVLlD]

[6] MIT Deep Learning:

- Convolutional Neural Network [https://www.youtube.com/watch?v=2xqkSUhmmXU]

[7] Super Data Science:

- Flatten Layer [https://shorturl.at/iBkNa]

[8] Adam W. Harley :

- CNN visualizer in browser [https://adamharley.com/nn_vis/cnn/3d.html]

[9] The Independent Code:

- CNN from scratch[https://www.youtube.com/watch?v=Lakz2MoHy6o&t=852s]

  (great video add-on for this paper)

[10] Spatial:

- Convolution [https://spatial-lang.org/conv]

[11] Mayank Kaushik on Medium:

- Convolution with Strides Part I [https://shorturl.at/GbckG]
- Convolution with Strides Part II [https://shorturl.at/yeWNr]

Recommend To Check Out:

CS231n CNN [https://shorturl.at/qvfc9]

The Independent Code [https://www.youtube.com/@independentcode/videos]

Author Contact Info:

venchislavcodes@gmail.com

https://github.com/Venchislaw

2024